Report - Praxisblatt 1

Malte A. Weyrich
Jan P. Hummel

7. Mai 2024

1 Aufgabe

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

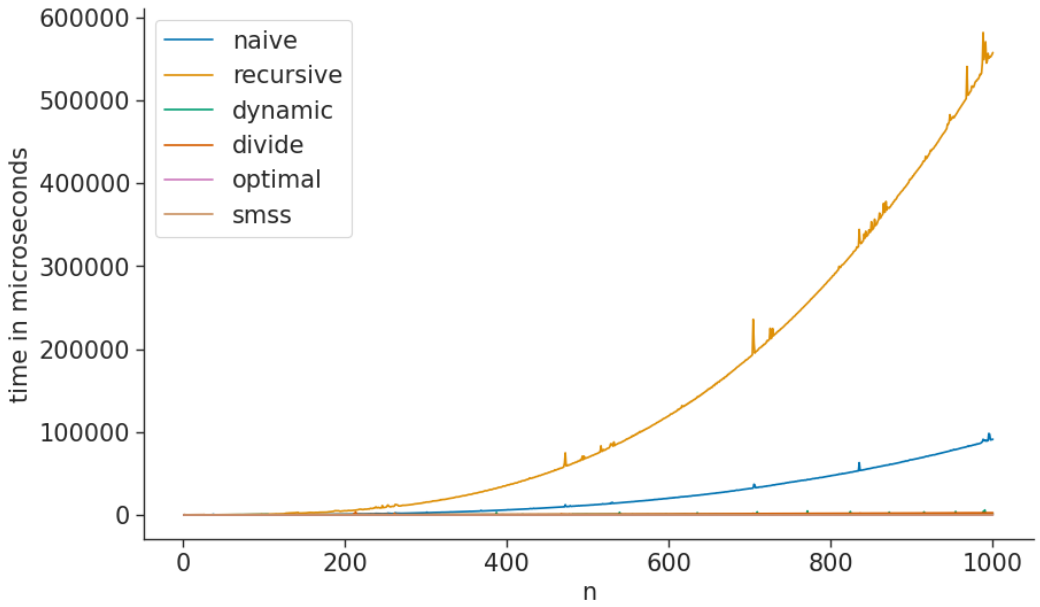

Abbildung 1: all

2 Aufgabe

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.1 2a - SMSS

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.2 2b - SMSS mit minimaler Länge

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.3 2c - Alle MSS

Für die 2c haben wir unseren ursprünglichen Ansatz nicht weiter verwendet. Der *optimale Algorithmus* ist in sich selbst so effizient, dass er null Folgen am Anfang und am Ende von Segmenten direkt ignoriert. Wir haben uns also überlegt, welcher der anderen Algorithmen mit eine möglichst guten Laufzeit, diese Fälle auch betrachtet. Wir haben uns dazu entschieden den *dynamic programming Algorithmus* zu erweitern. Dieser hat nämlich den Vorteil bereits ohne Erweiterungen, alle Kombinationen von Teilintervallen zu betrachten. Insbesondere die Teilintervalle, die der *optimale Algorithmus* nicht in Betracht zieht.

Beispiel

```
                          5 -5 9 -3 13 -5 5 -5 5
optimal              :    -----+=====+----------

tatsächliche MSS     :    -----+=====+----------
                          -----+==========+-----
                          -----+===============+
                          +==========+----------
                          +===============+-----
                          +=====================+
```

Der *dynamic programming Algorithmus* betrachtet in dem oberen Beispiel alle der Teilfolgen. Also haben wir wieder wie in 2.1 eine äußere Liste mit inneren Listen erstellt, und für jeden neuen *maxscore* eine neue innere Liste erstellt, die dann mit allen gleichwertigen Segmenten (also allen Segmenten mit demselben *maxscore*) befüllt wird, bis ein größerer *maxscore* gefunden wird. So findet der *2_c Algorithmus* also definitiv alle MSS, denn wir wissen bereits, dass der *dynamic programming* Ansatz korrekt ist und alle Kombinationen in Betracht zieht. Jedoch bußt dieser Ansatz bei der Speicherkomplexität ein, das *int[][]* Array wächst zu einem gegebenen Eingabe Vektor mit Länge n exponentiell: n^2. D.h. für große Eingaben verbrauchen wir enorm viel Speicherplatz.

Wie bereits in *1. Theorie Blatt Aufgabe 2* gezeigt, benutzt der *dynamic programming Algorithmus* immer nur die Hälfte des Arrays. Allgemein hat das Array die folgende Form:

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & 0 & & \times & \times \\ & & & & \times \end{pmatrix}$$

Man könnte also Platz einsparen, wo nur *0er* stehen:

Beispiel
Betrachten wir die jeweiligen Endzustände der Datenstrukturen von Algorithmus *2_c* und der Arbeitsspeicher schonenden Variante *2_c_1*:

- Eingabe: $v = \{5, -5, 9, -3, 13, -5, 5, -5, 5\}$
- Hardware: (*Processor: AMD® Ryzen 7 pro 4750u with radeon graphics × 16; Memory: 16.0 GB*)

```
Array S[n][n] of Default Dynamic Programming (2_c) on v:
   0: [5 ,0 ,9 ,6 ,19,14,19,14,19]
   1: [0 ,-5,4 ,1 ,14,9 ,14,9 ,14]
   2: [0 ,0 ,9 ,6 ,19,14,19,14,19]
   3: [0 ,0 ,0 ,-3,10,5 ,10,5 ,10]
   4: [0 ,0 ,0 ,0 ,13,8 ,13,8 ,13]
   5: [0 ,0 ,0 ,0 ,0 ,-5,0 ,-5, 0]
   6: [0 ,0 ,0 ,0 ,0 ,0 ,5 , 0, 5]
   7: [0 ,0 ,0 ,0 ,0 ,0 ,0 ,-5,0 ]
   8: [0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ]

Max n on given Hardware: 30808:
java -jar ... --algorithms 2_c --size 30808

ArrayList S containing int[] Arrays of Improved Dynamic Programming (2_c_1) on v:
   0: [5 ,0 ,9 ,6 ,19,14,19,14,19]
   1: [-5,4 ,1 ,14,9 ,14,9 ,14]
   2: [9 ,6 ,19,14,19,14,19]
   3: [-3,10,5 ,10,5 ,10]
   4: [13,8 ,13,8 ,13]
   5: [-5,0 ,-5,0]
   6: [5 ,0 ,5]
   7: [-5,0]
   8: [5]

Max n on given Hardware: 44069:
java -jar ... --algorithms 2_c_1 --size 44069
```

Der *improved dynamic programming 2_c_1* hat keine Position in der Datenstruktur, die ungenutzt bleibt. Der verbesserte Ansatz schafft es 13261 zusätzliche Zahlen zu verarbeiten, also ein $\approx 43\%$ größeres n als der *default 2_c Algorithmus* (*auf der gegebenen Hardware). An der Komplexität hat sich durch die Abänderungen nichts geändert. In Abbildung 2 werden die zwei Ansätze in ihrer Laufzeit verglichen. Die Spikes in der Abbildung entstehen durch die inkonsistente Geschwindigkeit bei der Array Initialisierung.

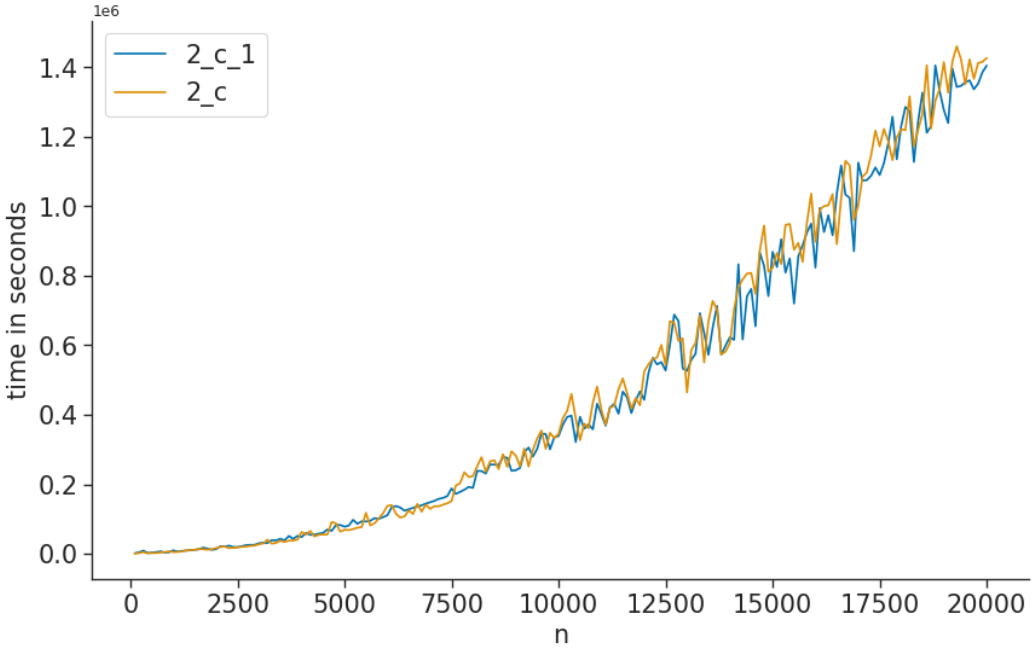

Abbildung 2: Ansatz 2_c verglichen mit dem Arbeitsspeicher optimierten Ansatz 2_c_1

2.4 Ergebnisse

Um die Ergebnisse auf dem Beispeil des Übungsblattes zu rekreiren ruft man die *JAR* folgendermaßen auf:

```
java -jar JAR --v 5 -2 5 -2 1 -9 12 -2 24 -5 13 -12 3 -13 5 -2 -1 2

// Naive:
//
//  [6,10] mit score 42
//  466 µs
//  for input size 19

// Recursive:
//
//  [6,10] mit score 42
//  180 µs
//  for input size 19

// Dynamic Programming:
//
//  [6,10] mit score 42
//  29 µs
//  for input size 19

// Divide and Conquer:
//
//  [8,8] mit score 24
//  847 µs
//  for input size 19

// Optimal:
//  [6,10] mit score 42
```

```
//  4 µs
//  for input size 19

// 2_a (MSS):
//
//  [6,10] mit score 42
//  24 µs
//  for input size 19

// 2_b (SMSS):
//
//  [6,10] mit score 42
//  24 µs
//  for input size 19

// 2_c (All SMSS):
//
//  [6,10] mit score 42
//  32 µs
//  for input size 19

// 2_c_1 (All SMSS & Optimized space usage):
//
//  [6,10] mit score 42
//  49 µs
//  for input size 19
```